Exhibit G-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Hitz et al.		
U.S. Patent No.	6,892,211	Issued: May 10, 2005	
Filed:	April 12, 2004		
Control Number:	95/000,328	Docket No.:	347155-29

Title: COPY ON WRITE FILE SYSTEM CONSISTENCY AND BLOCK USAGE

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage Via Express Mail No. EV 978 428 091 US in an envelope addressed to: Central Reexamination Unit (CRU), ATTN: "Box Inter Partes Reexam", Commissioner for Patents, United States Patent and Trademark Office, P.O. Box 1450, Alexandria, VA. 22313-1450 on:
January 14, 2008

Ronald L. Yin

Ronald L. Yin

* * *

RESPONSE TO NOTICE OF FAILURE TO COMPLY WITH INTER PARTES REEXAMINATION REQUEST FILING REQUIREMENTS (37 CFR 1.915(D))

Central Reexamination Unit (CRU)
ATTN: "Box Inter Partes Reexam"
Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA. 22313-1450

Sir:

In response to the notice mailed December 31, 2007 ("Notice"), Requester has attached herewith a Replacement Attachment for Re-Examination complying with 37 CFR 1.915. Specifically, in the Notice, the PTO stated that the request failed to comply with the requirements of 37 CFR 1.915(b)(3) because the phrase "at least" does not limit the number of claims to which each reference is applied and the phrase does not limit the number of claims to each identified substantial new question of patentability. This language has been eliminated.

In addition, the Notice alleged that the references: 1) Rosenblum and Ousterhout; 2) Kent; 3) Belsan; 4) Noveck; and 5) Gray set forth in the IDS accompanying the request were not explained in sufficient detail as provided by the regulations. This has also been rectified by the accompanying Replacement Attachment.

A copy of this response as well as the Replacement Attachment is served on the patent owner as provided in 37 CFR 1.33(c). The name and address of the party served and the date of service are:

Swernofsky Law Group PC
P.O. Box 390013
Mountain View, CA 94039

Served On: January 14, 2008

Any fee due for this reexamination may be charged to Deposit Account No. 07-1896.

Respectfully submitted,

DLA PIPER US LLP

Date: January 14, 2008

By: Ronald L. Yin

Ronald L. Yin
Reg. No. 27,607

Attorneys for Applicant(s)

Ronald L. Yin
DLA Piper US LLP
2000 University Avenue
East Palo Alto, CA 94303-2248
650-833-2437 (Direct)
650-833-2000 (Main)
650-833-2001 (Facsimile)
ronald.yin@dlapiper.com

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Hitz et al.		
U.S. Patent No.	6,892,211	Issued: May 10, 2005	
Filed:	April 12, 2004		
Control Number:	95/000,328	Docket No.:	347155-29

Title: COPY ON WRITE FILE SYSTEM CONSISTENCY AND BLOCK USAGE

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage Via Express Mail No. EV 978 428 091 US in an envelope addressed to: Central Reexamination Unit (CRU), ATTN: "Box Inter Partes Reexam", Commissioner for Patents, United States Patent and Trademark Office, P.O. Box 1450, Alexandria, VA. 22313-1450 on:
January 14, 2008

Ronald L. Yin

Ronald L. Yin

* * *

REPLACEMENT ATTACHMENT TO REQUEST FOR INTER-PARTES RE-EXAMINATION OF U.S. PATENT NO. 6,892,211

Central Reexamination Unit (CRU)
ATTN: "Box Inter Partes Reexam"
Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA. 22313-1450

Sir:

Pursuant to 35 U.S.C. §§ 311-318 and 37 CFR § 1.903-1.997, this is a request for inter-partes reexamination of United States Patent No. 6,892,211 which issued on May 10, 2005 to Hitz et al. (the "'211 Patent").

I. CLAIMS FOR WHICH REEXAMINATION IS REQUESTED

Reexamination is requested of Claims 1-24 of the '211 Patent in view of the prior art listed on the Citation of Prior Art under 37 CFR § 1.501 and 35 U.S.C. § 301 which is submitted with the Request for Reexamination.

II. EXPLANATION OF PERTINENCE AND MANNER OF APPLYING CITED PRIOR ART TO EVERY CLAIM FOR WHICH REEXAMINATION IS REQUESTED

Introduction

One or more prior art references submitted with the Citation of Prior Art renders claims 1-24 of the '211 Patent either anticipated under 35 USC 102(a), (b) or (e) or unpatentable under 35 USC 103 so that substantial new questions of patentability of the claims of the '211 Patent have been raised by this request for reexamination. For each claim for which reexamination is sought, a specific citation of the prior art or the combination of the prior art pertinent to the claim and a description of the relevancy of that prior art to the claim are set forth blow in greater detail. None of the prior art cited in the Citation of Prior Art was submitted to the examiner or considered by the examiner during the prosecution of the '211 Patent.

Statements Identifying Substantial New Questions of Patentability

The following Substantial New Question (SNQ) of Patentability are raised with respect to claims 1-24 of the '211 Patent in view of the prior art listed on the Citation of Prior Art.

SNQ 1. A substantial new question of patentability as to claims 1-24 is raised by the reference Quinlan.

SNQ 2. A substantial new question of patentability as to claims 1-24 is raised by the combination of the references Popek and Ylonen.

SNQ 3. A substantial new question of patentability as to claims 1-6, 9-14, 17-22 is raised by the references Seltzer.

SNQ 4. A substantial new question of patentability as to claims 1-24 raised by the reference Schilling.

SNQ 5. A substantial new question of patentability as to claims 1-4, 9-12, 17-20 is raised by the combination of the references Leffler and Bach.

SNQ 6. A substantial new question of patentability as to claims 1-24 is raised by the combination of the references Rosenblum, Ylonen and Leffler.

SNQ 7. A substantial new question of patentability as to claims 1- 3, 5-11, 13-19, 21-24 is raised by the references Kent, Popek and Ylonen.

SNQ 8. A substantial new question of patentability as to claims 1-6, 9-14, 17-22 is raised by the references Rosenblum and Ousterhoot in view of Leffler.

SNQ 9. A substantial new question of patentability as to claims 1, 9, and 17 is raised by the reference Belsan.

SNQ 10. A substantial new question of patentability as to claims 1, 9, and 17 is raised by the reference Noveck.

SNQ 11. A substantial new question of patentability as to claims 1-6, 9-14, 17-22 is raised by the reference Gray.

Explanation of How Each SNQ Is Raised

1. The Quinlan reference teaches a file system stored in a magnetic disc cache memory and on a WORM optical disk storage system. Quinlan further teaches a file system with a Unix file hierarchy. Quinlan, p. 1291. A root inode with levels of indirection, pointing directly and indirectly to data blocks, is inherent in a Unix file hierarchy. Figure 3 on p. 1295 further teaches a root inode directly and indirectly pointing to blocks. The root block (root inode) "contains the directory entry for the root of the tree and all blocks can trace a path from this root." Quinlan, p. 1291. Quinlan teaches the use of a cache maintaining in-core copies of a subset of the file system. Quinlan, p. 1290. Quinlan explains that a copy-on-write technique is used. Quinlan, pp. 1291-92. After a block is modified via copy-on-write in memory, its directory (inode) is also modified with a pointer to the new location. *Id.* For the system to operate, it is inherent that the root inode is cached in memory and modified as data is modified. Periodically, the Quinlan file system atomically advances from one consistent state to another by flushing the contents of the cache to disk, thereby forming a snapshot. Quinlan, p. 1292-95, Fig. 3. The active file system shares unmodified data blocks with the previous snapshots. Quinlan, p. 1292. "In effect, the snapshot and the file system will split apart as the file system is modified." Quinlan, p. 1294. After a snapshot is taken, the in-core root inode pointing to buffers and new blocks reflects changes from the previous consistent state: "The root of the file system is moved to a write

block. Note that changes to the file system are only visible through the new root block; the file system remains unchanged when viewed through the old root." *Id.* This is illustrated in Fig. 3. Thus, this teaching of Quinlan raises an SNQ with respect to at least claim 1 of the '211 Patent. A detailed explanation of the application of this reference to this SNQ with respect to other claims is set forth herein below under the First Basis of Invalidity.

2. The Popek and Ylonen references teach a LOCUS file system storing data in a memory and on hard disk. *See e.g.* Popek, Sec. 3.3, pp. 33-34, Sec. 3.4.6, pp. 46-48. Popek further teaches that LOCUS maintains an on-disk inode that contains "page numbers" or pointers to data blocks or to "intermediate node[s]" that point to data blocks Popek, Sec. 3.3, pp. 33-34, Sec. 3.4.6, pp. 46-48. "A file is composed of an inode and an associated ordered collection of data blocks." *Id.* at 34. "Additional mechanism is also present to support large files that are structured through indirect pages that contain page pointers." Popek, Sec. 3.4.6, p. 47. The logical pages or data blocks taught in Popek inherently represent a consistent state of the LOCUS file system. Finally, Popek teaches that LOCUS maintains an "in-core copy of the disk inode." Popek, Sec. 3.4.6, p. 47. LOCUS uses a shadow paging mechanism (copy-on-write) to update files without overwriting data in place. *Id.* As logical pages (logical blocks) are updated, "it is necessary to keep track of where the old and the new pages are. The disk inode contains the old page numbers. The in-core copy of the disk inode starts with the old pages, but is updated with new page numbers as shadow pages are allocated." *Id.* As logical pages are updated, they are periodically flushed to a new location on disk. *Id.* As a result, the in-core inode points to a set of blocks on disk that includes the old unchanged blocks and newly flushed blocks containing changes, as well as to memory buffers that have not yet been flushed. The sum total of these buffers and blocks inherently represents a second consistent state of the LOCUS file system. Popek also teaches that "[t]he important concept of atomically committing changes has been imported from the database world and integrated into LOCUS." Popek, Sec. 3.4.6, p. 46. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Ylonen. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time

the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed." *Id.* Thus, this teaching of Popek and Ylonen raises an SNQ with respect to at least claim 5 of the '211 Patent. A detailed explanation of application of these references to this SNQ with respect to other claims is set forth herein below under the Second Basis of Invalidity.

3. The Seltzer reference discloses an LFS file system, with data stored in memory and on a disk storage system. *See e.g.* Seltzer, p. 59. Seltzer also teaches that the LFS file system maintains an on-disk root inode that points directly and indirectly to a first set of storage blocks on disk in a consistent state. *See e.g.* Sec. 5.2.1.3 at p. 59, Sec. 6.1.1 at pp.70-71. This is further illustrated in Fig. 5-5. Seltzer teaches an in-memory inode containing the physical representation of the file system layout. Seltzer, Sec. 5.2.1.3, p. 59, Fig. 5-5. The in-memory inode "additionally includes lists of buffers and links to other inodes. . . . The in-memory inode is extended to have a list of transaction-protected buffers in addition to its clean and dirty buffer lists." *Id.* "When a file is written, the new data blocks are appended to the log, and the index structure and indirect blocks are modified (in memory) to contain the new disk address of the newly written block." Seltzer, Sec. 4.1, pp. 31-32. Therefore, Seltzer teaches that the in-memory inode represents a consistent up-to-date state of the file system, pointing directly and indirectly to buffers containing new or modified data as well to the unchanged blocks of data. Seltzer further teaches that "[w]hen writing, LFS gathers many dirty pages and prepares to write them to disk sequentially in the next available segment. At that time, LFS sorts the blocks by logical block number, assigns them disk addresses, and updates the meta-data to reflect their addresses." Seltzer, Sec. 6.1.1, p. 71. Once some buffers holding dirty pages (changed data) are flushed to disk blocks, they, together with unchanged blocks extant on disk, constitute a second set of blocks. This second set of blocks, in combination with un-flushed dirty buffers, represent the up-to-date file system in a consistent state. Thus, this teaching of Seltzer raises an SNQ with respect to at least claim 1 of the '211 Patent. A detailed explanation of the application of this reference to this SNQ with respect to other claims is set forth herein below under the Third Basis of Invalidity.

4. The Schilling reference discloses a WoFS file system using a cache memory and one or more WORM optical disks. Schilling further teaches that the WoFS file system can be used with

magnetic hard disks. Schilling, Sec. 1.2, p. 5. Schilling teaches that the WoFS file system is based on the well-known Unix file structure (as implemented in industry-standard BSD file system), which includes a superblock (root inode), inodes with levels of indirection, and data blocks. Schilling, Sec. 1.2.2, p. 6. WoFS modifies this structure to account for the write once – read many (WORM) nature of the optical media. *See e.g.* Schilling, Sec. 12.3-1.2.4. The modified system retains superblocks, inodes (referred to as *generation* nodes or *gnodes*) and data blocks. *Id.* These structures are present on the WORM disks. *See* Schilling Sec. 1.2.5. WoFS uses copy-on-write techniques to achieve file system consistency, writing new versions of files to a previously unused area of the medium. *See e.g.* Schilling, Sec. 1.2.5.1, p. 12. Schilling teaches a cache memory where an in-core copy of the on-disk file system structure is maintained. *See* Schilling, Sec. 1.4.2. Changes to the file system occur in the cached file system structure, which thereby diverges from the on-disk structure. *See* Schilling, Sec. 1.2.7. It is inherent that changed data is stored in buffers in the cache before being written to disk. Schilling teaches that gnode updates are periodically forced to disk. Schilling, Sec. 1.2.7.4, p. 20. It is inherent that updated gnodes are buffered in the cache before being written to disk. Schilling further teaches that the superblock is written to disk after data blocks and intermediate gnodes. Schilling, Sec. 1.2.7.2, p. 19. It is inherent that the updated superblock, pointing to buffered and on-disk gnodes and data blocks, is buffered in the cache before being forced to disk. Thus, this teaching of Schilling raises an SNQ with respect to at least claim 1 of the '211 Patent. A detailed explanation of the application of this reference to this SNQ with respect to other claims is set forth herein below under the Fourth Basis of Invalidity.

5. The Leffler and Bach references disclose an implementation of the well-known Unix file system, storing information in memory and on hard disk. *See* Leffler, Ch. 7. Leffler teaches maintaining an on-disk root inode pointing directly and indirectly to a set of blocks on disk in a consistent state, *See* Leffler Sec. 7.2 and Figure 7.6. Leffler teaches that a copy of the root inode (inode describing the root directory) and copies of lower level inodes and data blocks are maintained in memory (in-core). Leffler, Sec. 7.4, pp. 203-07 When the system writes new data, it allocates buffers in memory for the new data. *See e.g.* Leffler, Sec. 7.5, pp. 211-12. Periodically, the buffers are flushed to disk during a *sync* process. *See e.g.* Leffler, Sec. 7.4, p. 207. It is inherent that the incore root inode is updated to point to the new buffers and blocks. For instance, Bach (a conventional well known Unix textbook) teaches that in Unix, the in-core

inode is updated when data blocks change. Bach, Sec. 4.1, pp. 62-63. Thus, this teaching of the combination of Leffler and Bach raises an SNQ with respect to at least claim 1 of the '211 Patent. A detailed explanation of the application of these references to this SNQ with respect to other claims is set forth herein below under the Fifth Basis of Invalidity.

6. The Rosenblum, Ylonen and Leffler references disclose a file system stored in memory and on hard disks (see Rosenblum). Rosenblum maintains an on disk root inode. The format of the inodes and indirect blocks is the same as in the standard Unix file system, which is known to one of ordinary skill to provide for root inodes that directly and indirectly point to data blocks. *See* Rosenblum, Sec. 4.2.1, p.6. Rosenblum teaches that file system metadata, including inodes, and data blocks are cached in memory, where changes are accumulated before being forced to disk. Rosenblum, Sec. 4.1, p. 5. Before being forced to disk, the modified in-core root inode inherently points to buffers in memory containing modified data, new blocks already forced to disk, and old blocks with unmodified data. Thus, this teaching of Rosenblum raises an SNQ with respect to at least claim 1 of the '211 Patent. A detailed explanation of the application of the combination of Rosenblum, Ylonen and Leffler to this SNQ with respect to other claims is set forth herein below under the Sixth Basis of Invalidity.

7. Kent discloses a database using page shadowing (copy-on-write) and atomic commit to advance the database from one consistent state to another. Kent, Sec. 3.5, pp. 28-31, 40-43. The Kent database inherently resides on one or more hard disks. As is understood by one of ordinary skill in the art, databases and file systems are closely related fields of art. For example, Popek teaches that "[t]he important concept of atomically committing changes has been imported from the database world and integrated into LOCUS." Popek, Sec. 3.4.6, p. 46. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Kent and Ylonen to implement a file system. Kent database structures include the root, which points to the global version of each mapping page: "That is, the root defines the most recent, consistent state of the page table (Just as the page table defined the most recent, consistent state of the logical database.)" Kent, p. 41, Fig. 3-4. The structures shown in Fig. 3-4 would be understood by one of ordinary skill in the art to be equivalent to root inode, intermediate inodes, and data blocks in a file system. These structures reside on disk and in-core in cache memory. See e.g. Kent, pp. 28-31, 89-90. The cache memory in Kent contains buffers that hold modified data blocks. Kent, p. 90. Kent describes a number of "primitives" – operations that allow the

shadowing algorithm CRMshadow to "maintain[] buffers and . . . flush[] them to disk." Kent, Sec. 3.5.2, p. 34. The structure shown in Fig. 3-4 is held in-core, with the root pointing directly and indirectly to buffers storing metadata (such as page tables) and data. Before it is flushed to disk, the root also points to updated blocks on disk. Kent, Sec. 3.5.2, pp. 40-43. Kent teaches an atomic commit operation by flushing the root from cache memory to disk: "The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root." Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47. Thus this teaching of Kent, Popek and Ylonen raises an SNQ with respect to at least claim 1 of the '211 Patent. A detailed explanation of the application of the combination of Kent, Popek and Ylonen to this SNQ with respect to other claims is set forth herein below under the Seventh Basis of Invalidity.

8. Rosenblum and Ousterhout teaches a file system stored in memory and on hard disks. Rosenblum and Ousterhout maintains an on disk root inode. The format of the inodes and indirect blocks is the same as in the standard Unix file system, which is known to one of ordinary skill to provide for root inodes that directly and indirectly point to data blocks. See Rosenblum and Ousterhout, Sec. 3.1, p. 6, see also e.g. Leffler, Fig. 7.6, p. 194. Rosenblum and Ousterhout teaches that file system metadata, including inodes, and data blocks are cached in memory, where changes are accumulated before being forced to disk. Rosenblum and Ousterhout, Sec. 3, p. 3. Before being forced to disk, the modified in-core root inode inherently points to buffers in memory containing modified data, modified blocks, and old blocks with unmodified data. Thus this teaching of Rosenblum and Ousterhout and Leffler raises an SNQ with respect to at least claim 1 of the '211 Patent. A detailed explanation of the application of the combination of Rosenblum and Ousterhout and Leffler to this SNQ with respect to other claims is set forth herein below under the Eighth Basis of Invalidity.

9. Fig. 1 of Belsan discloses that the file system runs on a processor 101 and includes a cache memory 113. More specifically, Fig. 4 shows processor 204-0 and cache memory 113. Col. 6:2-27. Belsan Fig. 1 discloses a storage system including one or more hard disks 103-1 (disk drive subset). Data storage subsystem 100 is a file system. Col. 3:34-36. Belsan teaches a mapping table and a copy table. The mapping table (root inode) is maintained on disk and contains pointers to data blocks stored on disk. Col. 7:46-66. The mapping table is stored (backed up) on disk. Col. 6:36-39. Mapping tables store a plurality of pointers identifying data

blocks. Col. 8:64-9:27, Figs. 2 and 3. Belsan further teaches that the mapping table is uploaded and modified in cache memory during write transactions, constituting an incore root inode. Contents of the on-disk mapping table are loaded into a cache hash table. See e.g. Col. 13:54-59. The disclosed file system uses copy-on-write, so that a block to be written is first copied into cache memory and then is re-written to a new location on disk. Col. 12:22-38; 13:42-14:6. Further, during a sequence of write operations, the cached mapping table and associated copy table point to some data buffers in cache (buffers that are currently being written) as well as some modified blocks The cached mapping table also points to unmodified blocks on disk, which are also pointed to by the un-updated mapping table stored on-disk. Thus this teaching of Belsan raises an SNQ with respect to at least claim 9 of the '211 Patent. A detailed explanation of the application of Belsan to this SNQ with respect to other claims is set forth herein below under the Ninth Basis of Invalidity.

10. Noveck teaches a server including one or more hard disks, a cache memory and maintaining a file system. See Fig.1, Col. 6:20-24, Col. 7:14-17. Instructions for maintaining the Noveck file system are inherently stored on a machine-readable medium. Noveck discloses an on-disk Unix inode, with direct and indirect blocks. Col. 3:29-35, 3:60-63. Noveck further teaches an in-core inode, which is an image of the on-disk inode, maintained in cache memory. Col. 6:25-31. As data in the file system is modified, the in-core structure differs from the on-disk inode and data. Col. 7: 14-24. The in-core inode points to blocks and buffers that have been modified since the on-disk inode was written. Thus this teaching of Noveck raises an SNQ with respect to at least claim 17 of the '211 Patent. A detailed explanation of the application of Noveck to this SNQ with respect to other claims is set forth herein below under the Tenth Basis of Invalidity.

11. Gray teaches a database system using shadowing (copy-on-write) to manage files. See e.g. Gray, Sec. 2.1, Fig. 7. The database in Gray inherently resides in memory and on disk. A copy of a page table (equivalent to an inode) and data pages (equivalent to data blocks) as well as a directory root (equivalent to root inode) is cached in memory and resident on disk. Gray, Sec. 2.1, Sec. 2.7, Fig. 10. A copy of a page table (equivalent to an inode) and data pages (equivalent to data blocks) as well as a directory root (equivalent to root inode) is cached in memory and resident on disk. Gray, Sec. 2.1, Sec. 2.7, Fig. 10. Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing

to disk an updated directory root. Gray, Sec. 2.7, Fig. 10. It is inherent that the in-memory copy of the directory root points to buffers and blocks representing the database in a present consistent state. Changes between the present consistent state and the checkpointed previous state are inherently stored in buffers and blocks. Thus this teaching of Gray raises an SNQ with respect to at least claim 1 of the '211 Patent. A detailed explanation of the application of Gray to this SNQ with respect to other claims is set forth herein below under the Eleventh Basis of Invalidity.

Invalidity

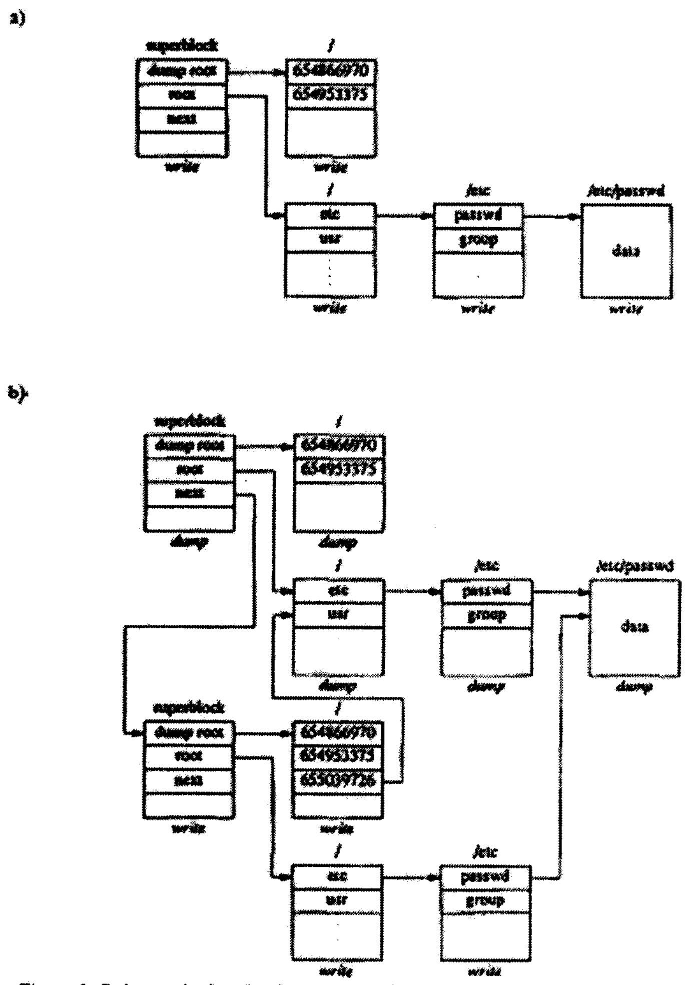
First Basis of Invalidity

The reference applicable to the first basis of invalidity is:

1. Quinlan, *A Cached WORM File System*, Software – Practice And Experience, Vol. 21(12), 1289-1299, December 1991 ("Quinlan").

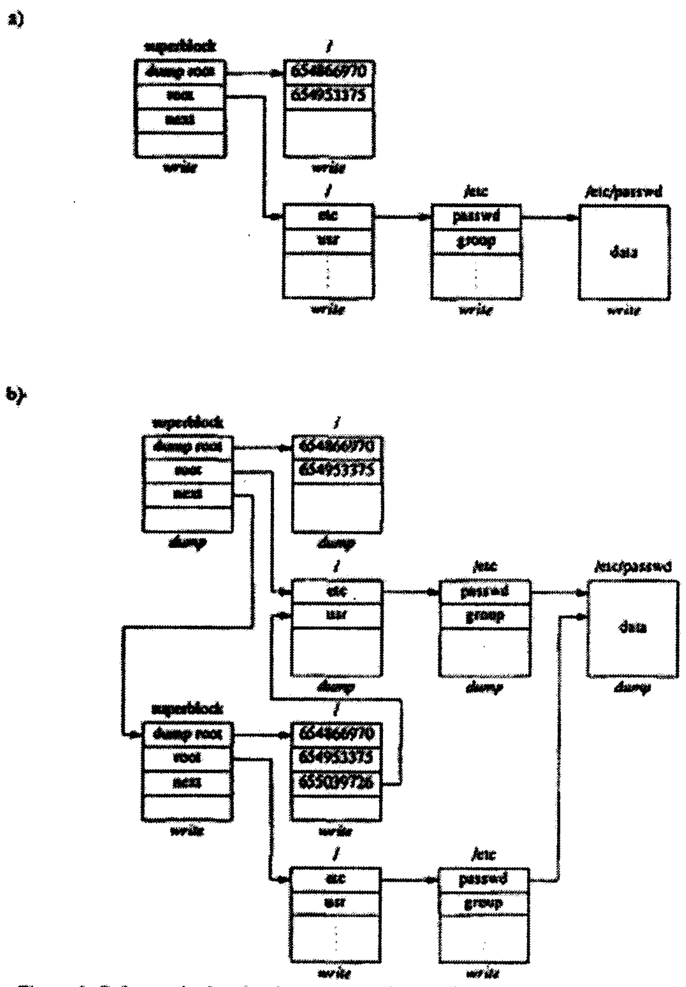
The pertinence and manner of applying Quinlan to claims 1-24 for which re-examination is requested is as follows:

Claims of '211 Patent	Quinlan
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Quinlan teaches a file system stored in a magnetic disc cache memory and on a WORM optical disk storage system.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	Quinlan teaches a file system with a Unix file hierarchy. Quinlan, p. 1291. A root inode with levels of indirection, pointing directly and indirectly to data blocks, is inherent in a Unix file hierarchy. Figure 3 on p. 1295 further teaches a root inode directly and indirectly pointing to blocks. The root block (root inode) "contains the directory entry for the root of the tree and all blocks can trace a path from this root." Quinlan, p. 1291.
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a	Quinlan teaches the use of a cache maintaining in-core copies of a subset of the file system. Quinlan, p. 1290. Quinlan explains that a copy-on-write technique is used. Quinlan, pp. 1291-92. After a block is modified via copy-on-write in memory, its directory (inode) is also modified with a pointer to the new location. *Id.* For the system to operate, it is inherent that the root inode is

second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	cached in memory and modified as data is modified. Periodically, the Quinlan file system atomically advances from one consistent state to another by flushing the contents of the cache to disk, thereby forming a snapshot. Quinlan, p. 1292-95, Fig. 3. The active file system shares unmodified data blocks with the previous snapshots. Quinlan, p. 1292. "In effect, the snapshot and the file system will split apart as the file system is modified." Quinlan, p1294. After a snapshot is taken, the in-core root inode pointing to buffers and new blocks reflects changes from the previous consistent state: The root of the file system is moved to a write block. . . . Note that changes to the file system are only visible through the new root block; the file system remains unchanged when viewed through the old root." *Id.* This is illustrated in Fig. 3:  *Figure 3. Before and after the dump command, assuming /etc/passwd is open*
2. A method as in claim 1, wherein	Quinlan teaches that the system advances from one

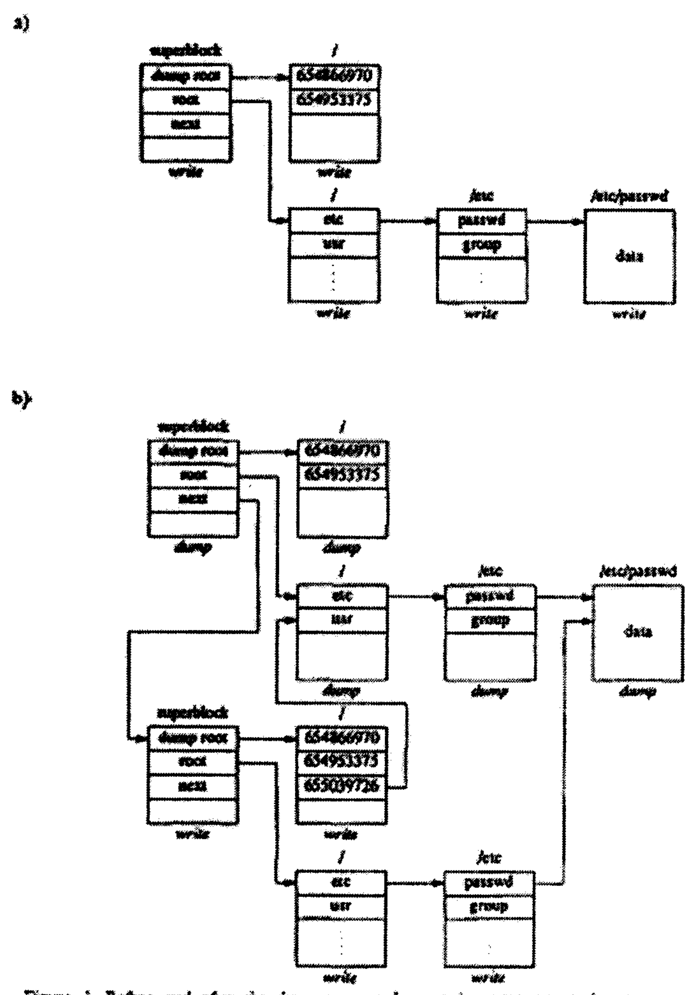
said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	consistent state to the next atomically, using the *dump* operation: "The dump is performed as an atomic operation." Quinlan, p. 1294.
3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Quinlan teaches that at the end of the *dump* operation the root of the file system (root inode) is updated so as to provide access to the snapshot. Quinlan, p. 1294. It is inherent that this operation occurs after all the other data is flushed to disk. This is also reflected in Fig. 3.
4. A method as in claim 3, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. In view of the knowledge in the art it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
5. A method as in claim 1, further comprising the step of creating a snapshot of said file system by copying only said on-disk root inode.	Quinlan teaches the creation of multiple snapshots. To do this, the root inode is updated so that it points to the most recent snapshot. Quinlan, p. 1294, Fig. 3.
6. A method as in claim 5, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Quinlan teaches that the efficiency of the file system is enhanced because "the number of blocks to write is minimal, as any block that has not changed since the previous dump will be shared in both snapshots and need not be written." Quinlan, p. 1295.
7. A method as in claim 1, further comprising the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	Quinlan teaches the creation of multiple snapshots. To do this, the root inode is updated so that it points to the most recent snapshot. Quinlan, p. 1294, Fig. 3.
8. A method as in claim 7, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage	The active file system shares unmodified data blocks with the previous snapshots. Quinlan, p. 1292. "In effect, the snapshot and the file system will split apart as the file system is modified." Quinlan, p1294. Quinlan further teaches that the efficiency of the file system is

system.	enhanced because "the number of blocks to write is minimal, as any block that has not changed since the previous dump will be shared in both snapshots and need not be written." Quinlan, p. 1295.
9. A device comprising: a processor; a memory; and	The file system taught in Quinlan is inherently executed on a processor with a memory.
a storage system including one or more hard disks;	Quinlan file system has data stored in cache memory and on a WORM disk storage system.
wherein said memory and said storage system store a file system; and	Quinlan file system has data stored in cache memory and on a WORM disk storage system.
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Quinlan teaches a file system with a Unix file hierarchy. Quinlan, p. 1291. A root inode with levels of indirection, pointing directly and indirectly to data blocks, is inherent in a Unix file hierarchy. Figure 3 on p. 1295 further teaches a root inode directly and indirectly pointing to blocks. The root block (root inode) "contains the directory entry for the root of the tree and all blocks can trace a path from this root." Quinlan, p. 1291. Quinlan further teaches the use of a cache maintaining in-core copies of a subset of the file system. Quinlan, p. 1290. Quinlan explains that a copy-on-write technique is used. Quinlan, pp. 1291-92. After a block is modified via copy-on-write in memory, its directory (inode) is also modified with a pointer to the new location. *Id.* For the system to operate, it is inherent that the root inode is cached in memory and modified as data is modified. Periodically, the Quinlan file system atomically advances from one consistent state to another by flushing the contents of the cache to disk, thereby forming a snapshot. Quinlan, p. 1292-95, Fig. 3. The active file system shares unmodified data blocks with the previous snapshots. Quinlan, p. 1292. "In effect, the snapshot and the file system will split apart as the file system is modified." Quinlan, p1294. After a snapshot is taken, the in-core root inode pointing to buffers and new blocks reflects changes from the previous consistent state: The root of the file system is moved to a write block. . . . Note that changes to the file system are only visible through the new root block; the file system remains unchanged when viewed through the old root." *Id.* This is illustrated in Fig. 3:

	 a) superblock dump root root next write / 654866970 654953375 write / etc usr write /etc passwd group write /etc/passwd data write b) superblock dump root root next dump / 654866970 654953375 dump / etc usr dump /etc passwd group dump /etc/passwd data dump superblock dump root root next write / 654866970 654953375 655039726 write / etc usr write /etc passwd group write Figure 3. Before and after the dump command, assuming /etc/passwd is open
10. A device as in claim 9, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Quinlan teaches that the system advances from one consistent state to the next atomically, using the *dump* operation: "The dump is performed as an atomic operation." Quinlan, p. 1294.
11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Quinlan teaches that at the end of the *dump* operation the root of the file system (root inode) is updated so as to provide access to the snapshot. Quinlan, p. 1294. It is inherent that this operation occurs after all the other data is flushed to disk. This is also reflected in Fig. 3.
12. A device as in claim 11, wherein updating said on-disk root inode	Replicating the metadata, such as a root inode, is a technique that is well known in the art. In view of the

further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	knowledge in the art it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
13. A device as in claim 9, wherein the instructions further comprise the step of creating a snapshot of said file system by copying only said on-disk root inode.	Quinlan teaches the creation of multiple snapshots. To do this, the root inode is updated so that it points to the most recent snapshot. Quinlan, p. 1294, Fig. 3.
14. A device as in claim 13, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Quinlan teaches that the efficiency of the file system is enhanced because "the number of blocks to write is minimal, as any block that has not changed since the previous dump will be shared in both snapshots and need not be written." Quinlan, p. 1295.
15. A device as in claim 9, wherein the instructions further comprise the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	Quinlan teaches the creation of multiple snapshots. To do this, the root inode is updated so that it points to the most recent snapshot. Quinlan, p. 1294, Fig. 3.
16. A device as in claim 15, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	The active file system shares unmodified data blocks with the previous snapshots. Quinlan, p. 1292. "In effect, the snapshot and the file system will split apart as the file system is modified." Quinlan, p1294. Quinlan further teaches that the efficiency of the file system is enhanced because "the number of blocks to write is minimal, as any block that has not changed since the previous dump will be shared in both snapshots and need not be written." Quinlan, p. 1295.
17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor,	The Quinlan file system is inherently stored on machine readable medium in the form of instructions executable by a processor coupled to a memory and one or more hard disks. Quinlan teaches a file system with a Unix file hierarchy. Quinlan, p. 1291. A root inode with levels of indirection, pointing directly and indirectly to data blocks, is inherent in a Unix file hierarchy. Figure 3 on p. 1295 further teaches a root inode directly and

cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	indirectly pointing to blocks. The root block (root inode) "contains the directory entry for the root of the tree and all blocks can trace a path from this root." Quinlan, p. 1291. Quinlan further teaches the use of a cache maintaining in-core copies of a subset of the file system. Quinlan, p. 1290. Quinlan explains that a copy-on-write technique is used. Quinlan, pp. 1291-92. After a block is modified via copy-on-write in memory, its directory (inode) is also modified with a pointer to the new location. *Id.* For the system to operate, it is inherent that the root inode is cached in memory and modified as data is modified. Periodically, the Quinlan file system atomically advances from one consistent state to another by flushing the contents of the cache to disk, thereby forming a snapshot. Quinlan, p. 1292-95, Fig. 3. The active file system shares unmodified data blocks with the previous snapshots. Quinlan, p. 1292. "In effect, the snapshot and the file system will split apart as the file system is modified." Quinlan, p1294. After a snapshot is taken, the in-core root inode pointing to buffers and new blocks reflects changes from the previous consistent state: The root of the file system is moved to a write block. . . . Note that changes to the file system are only visible through the new root block; the file system remains unchanged when viewed through the old root." *Id.* This is illustrated in Fig. 3:

	 a) superblock dump root root next write / 654866970 654953375 etc usr /etc passwd group /etc/passwd data b) dump 655039726 Figure 3. Before and after the dump command, assuming /etc/passwd is open
18. An article of manufacture as in claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second consistent state.	Quinlan teaches that the system advances from one consistent state to the next atomically, using the *dump* operation: "The dump is performed as an atomic operation." Quinlan, p. 1294.
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information	Quinlan teaches that at the end of the *dump* operation the root of the file system (root inode) is updated so as to provide access to the snapshot. Quinlan, p. 1294. It is inherent that this operation occurs after all the other data is flushed to disk. This is also reflected in Fig. 3.

from said incore root inode.	
20. An article of manufacture as in claim 19, wherein updating said on-disk root inode comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. In view of the knowledge in the art it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
21. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create a snapshot of said file system by copying only said on-disk root inode.	Quinlan teaches the creation of multiple snapshots. To do this, the root inode is updated so that it points to the most recent snapshot. Quinlan, p. 1294, Fig. 3.
22. An article of manufacture as in claim 21, wherein the instructions cause the processor to create said snapshot so that said snapshot and said file system share said first set of blocks on said storage system when said snapshot is created.	Quinlan teaches that the efficiency of the file system is enhanced because "the number of blocks to write is minimal, as any block that has not changed since the previous dump will be shared in both snapshots and need not be written." Quinlan, p. 1295.
23. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create plural snapshots of said file system by copying only said on-disk root inode at different times.	Quinlan teaches the creation of multiple snapshots. To do this, the root inode is updated so that it points to the most recent snapshot. Quinlan, p. 1294, Fig. 3.
24. An article of manufacture as in claim 23, wherein the instructions, when executed by the processor, cause the processor to create each one of said plural snapshots so that each one of said snapshots and said file system share said first set of blocks on said storage system when each one of said plural snapshots is created.	The active file system shares unmodified data blocks with the previous snapshots. Quinlan, p. 1292. "In effect, the snapshot and the file system will split apart as the file system is modified." Quinlan, p1294. Quinlan further teaches that the efficiency of the file system is enhanced because "the number of blocks to write is minimal, as any block that has not changed since the previous dump will be shared in both snapshots and need not be written." Quinlan, p. 1295.

Second Basis of Invalidity

The references applicable to the second basis of invalidity are:

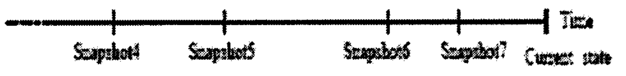
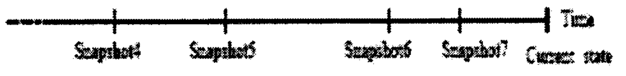
1. Popek, Walker, *The LOCUS Distributed System Architecture*, MIT Press, Cambridge, Mass., 1985 ("Popek").

2. Ylonen, *Concurrent Shadow Paging: A New Direction for Database Research*, Helsinki University of Technology, TKO-B86, 1992 ("Ylonen")

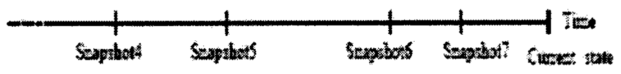
The pertinence and manner of applying Popek and Ylonen to claims 1-24 for which re-examination is requested is as follows:

Claims of '211 Patent	**Popek and Ylonen**
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Popek teaches a LOCUS file system storing data in a memory and on hard disk. *See e.g.* Popek, Sec. 3.3, pp. 33-34, Sec. 3.4.6, pp. 46-48.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	Popek teaches that LOCUS maintains an on-disk inode that contains "page numbers" or pointers to data blocks or to "intermediate node[s]" that point to data blocks Popek, Sec. 3.3, pp. 33-34, Sec. 3.4.6, pp. 46-48. "A file is composed of an inode and an associated ordered collection of data blocks." *Id.* at 34. "Additional mechanism is also present to support large files that are structured through indirect pages that contain page pointers." Popek, Sec. 3.4.6, p. 47. The logical pages or data blocks taught in Popek inherently represent a consistent state of the LOCUS file system.
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Popek teaches that LOCUS maintains an "in-core copy of the disk inode." Popek, Sec. 3.4.6, p. 47. LOCUS uses a shadow paging mechanism (copy-on-write) to update files without overwriting data in place. *Id.* As logical pages (logical blocks) are updated, "it is necessary to keep track of where the old and the new pages are. The disk inode contains the old page numbers. The in-core copy of the disk inode starts with the old pages, but is updated with new page numbers as shadow pages are allocated." *Id.* As logical pages are updated, they are periodically flushed to a new location on disk. *Id.* As a result, the in-core inode points to a set of blocks on disk that includes the old unchanged blocks and newly flushed blocks containing

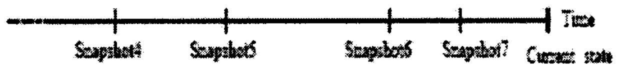
	changes, as well as to memory buffers that have not yet been flushed. The sum total of these buffers and blocks inherently represents a second consistent state of the LOCUS file system.
2. A method as in claim 1, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Popek teaches an "atomic commit operation" that moves the file system from one consistent state to the next. *See* Popek, Sec. 3.4.6, p. 47.
3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Popek teaches flushing changed logical pages to disk and then executing an atomic commit operation by "moving the in-core inode information to the disk inode." Popek, Sec. 3.4.6, p. 47.
4. A method as in claim 3, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. In view of the knowledge in the art, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
5. A method as in claim 1, further comprising the step of creating a snapshot of said file system by copying only said on-disk root inode.	Popek teaches that "[t]he important concept of atomically committing changes has been imported from the database world and integrated into LOCUS." Popek, Sec. 3.4.6, p. 46. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Ylonen. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have an arbitrary number of snapshots at different times, and they will all be

	consistent as long as their pages are not freed." *Id.*
6. A method as in claim 5, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Ylonen teaches a shadow paging (copy-on-write) technique that "never modifies accessible pages," therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.  Snapshot4 Snapshot5 Snapshot6 Snapshot7 Time Current state Figure 2: Snapshots represent consistent database states as of some time in the past.
7. A method as in claim 1, further comprising the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed." *Id.*
8. A method as in claim 7, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	Ylonen teaches a shadow paging (copy-on-write) technique that "never modifies accessible pages," therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.  Snapshot4 Snapshot5 Snapshot6 Snapshot7 Time Current state Figure 2: Snapshots represent consistent database states as of some time in the past.
9. A device comprising: a processor; a memory; and	The file system taught in Popek is inherently executed on a processor with a memory.
a storage system including one or more	Popek teaches a LOCUS file system storing data in

hard disks;	a memory and on hard disk. *See e.g.* Popek, Sec. 3.3, pp. 33-34, Sec. 3.4.6, pp. 46-48.
wherein said memory and said storage system store a file system; and	The file system in Popek is stored in memory and on hard disk. *Id.*
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Popek teaches that LOCUS maintains an on-disk inode that contains "page numbers" or pointers to data blocks or to "intermediate node[s]" that point to data blocks Popek, Sec. 3.3, pp. 33-34, Sec. 3.4.6, pp. 46-48. "A file is composed of an inode and an associated ordered collection of data blocks." *Id.* at 34. "Additional mechanism is also present to support large files that are structured through indirect pages that contain page pointers." Popek, Sec. 3.4.6, p. 47. The logical pages or data blocks taught in Popek inherently represent a consistent state of the LOCUS file system. Popek further teaches that LOCUS maintains an "in-core copy of the disk inode." Popek, Sec. 3.4.6, p. 47. LOCUS uses a shadow paging mechanism (copy-on-write) to update files without overwriting data in place. *Id.* As logical pages (logical blocks) are updated, "it is necessary to keep track of where the old and the new pages are. The disk inode contains the old page numbers. The in-core copy of the disk inode starts with the old pages, but is updated with new page numbers as shadow pages are allocated." *Id.* As logical pages are updated, they are periodically flushed to a new location on disk. *Id.* As a result, the in-core inode points to a set of blocks on disk that includes the old unchanged blocks and newly flushed blocks containing changes, as well as to memory buffers that have not yet been flushed. The sum total of these buffers and blocks inherently represents a second consistent state of the LOCUS file system.
10. A device as in claim 9, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Popek teaches an "atomic commit operation" that moves the file system from one consistent state to the next. *See* Popek, Sec. 3.4.6, p. 47.
11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers	Popek teaches flushing changed logical pages to disk and then executing an atomic commit operation by "moving the in-core inode information to the disk inode." Popek, Sec. 3.4.6,

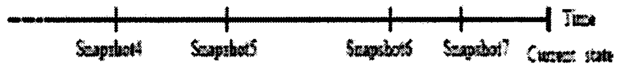
to said storage system before updating said on-disk root inode with information from said incore root inode.	p. 47.
12. A device as in claim 11, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. In view of the knowledge in the art, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
13. A device as in claim 9, wherein the instructions further comprise the step of creating a snapshot of said file system by copying only said on-disk root inode.	Popek teaches that "[t]he important concept of atomically committing changes has been imported from the database world and integrated into LOCUS." Popek, Sec. 3.4.6, p. 46. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Ylonen. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed." *Id.*
14. A device as in claim 13, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Ylonen teaches a shadow paging (copy-on-write) technique that "never modifies accessible pages," therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.  Figure 2: Snapshots represent consistent database states as of some time in the past.

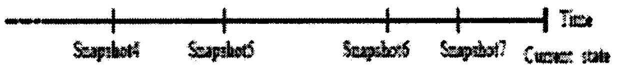
EM\7225346.3

15. A device as in claim 9, wherein the instructions further comprise the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed." *Id.*
16. A device as in claim 15, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	Ylonen teaches a shadow paging (copy-on-write) technique that "never modifies accessible pages," therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.  Time Snapshot4 Snapshot5 Snapshot6 Snapshot7 Current state Figure 2: Snapshots represent consistent database states as of some time in the past.
17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers	The file system of Popek is inherently stored on machine readable medium in the form of instructions executable by a processor coupled to a memory and one or more hard disks. Popek teaches that LOCUS maintains an on-disk inode that contains "page numbers" or pointers to data blocks or to "intermediate node[s]" that point to data blocks Popek, Sec. 3.3, pp. 33-34, Sec. 3.4.6, pp. 46-48. "A file is composed of an inode and an associated ordered collection of data blocks." *Id.* at 34. "Additional mechanism is also present to support large files that are structured through indirect pages that contain page pointers." Popek, Sec. 3.4.6, p. 47. The logical pages or data blocks taught in Popek inherently represent a consistent state of the LOCUS file system.

EM\7225346.3

in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Popek further teaches that LOCUS maintains an "in-core copy of the disk inode." Popek, Sec. 3.4.6, p. 47. LOCUS uses a shadow paging mechanism (copy-on-write) to update files without overwriting data in place. *Id.* As logical pages (logical blocks) are updated, "it is necessary to keep track of where the old and the new pages are. The disk inode contains the old page numbers. The in-core copy of the disk inode starts with the old pages, but is updated with new page numbers as shadow pages are allocated." *Id.* As logical pages are updated, they are periodically flushed to a new location on disk. *Id.* As a result, the in-core inode points to a set of blocks on disk that includes the old unchanged blocks and newly flushed blocks containing changes, as well as to memory buffers that have not yet been flushed. The sum total of these buffers and blocks inherently represents a second consistent state of the LOCUS file system.
18. An article of manufacture as in claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second consistent state.	Popek teaches an "atomic commit operation" that moves the file system from one consistent state to the next. *See* Popek, Sec. 3.4.6, p. 47.
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Popek teaches flushing changed logical pages to disk and then executing an atomic commit operation by "moving the in-core inode information to the disk inode." Popek, Sec. 3.4.6, p. 47.
20. An article of manufacture as in claim 19, wherein updating said on-disk root inode comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. In view of the knowledge in the art, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.

21. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create a snapshot of said file system by copying only said on-disk root inode.	Popek teaches that "[t]he important concept of atomically committing changes has been imported from the database world and integrated into LOCUS." Popek, Sec. 3.4.6, p. 46. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Ylonen. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed." *Id.*
22. An article of manufacture as in claim 21, wherein the instructions cause the processor to create said snapshot so that said snapshot and said file system share said first set of blocks on said storage system when said snapshot is created.	Ylonen teaches a shadow paging (copy-on-write) technique that "never modifies accessible pages," therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.  Time Snapshot4 Snapshot5 Snapshot6 Snapshot7 Current state Figure 2: Snapshots represent consistent database states as of some time in the past.
23. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create plural snapshots of said file system by copying only said on-disk root inode at different times.	Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have

	an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed." *Id.*
24. An article of manufacture as in claim 23, wherein the instructions, when executed by the processor, cause the processor to create each one of said plural snapshots so that each one of said snapshots and said file system share said first set of blocks on said storage system when each one of said plural snapshots is created.	Ylonen teaches a shadow paging (copy-on-write) technique that "never modifies accessible pages," therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.  Time Snapshot4 Snapshot5 Snapshot6 Snapshot7 Current state Figure 2: Snapshots represent consistent database states as of some time in the past.

Third Basis of Invalidity

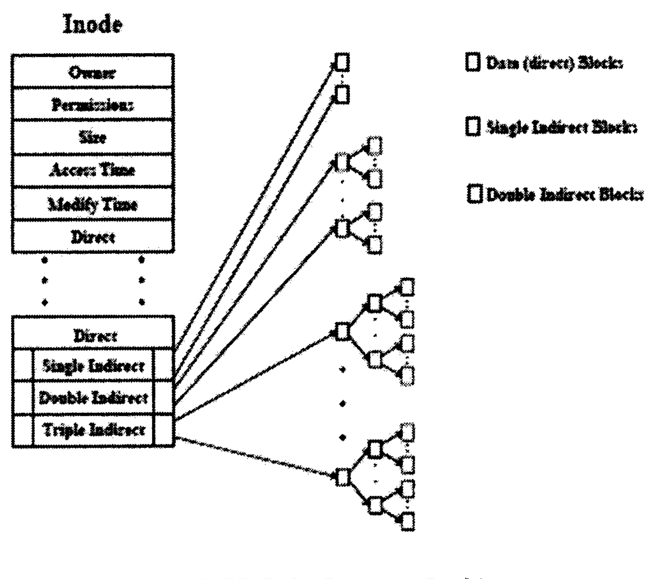
The reference applicable to the third basis of invalidity is:

1. Margo Ilene Seltzer , *File System Performance and Transaction Support*, Doctoral Dissertation, UC Berkeley, 1992 ("Seltzer").

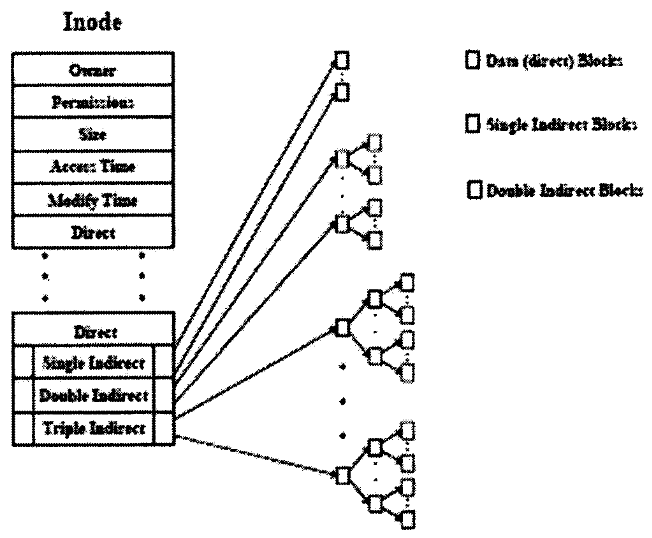
The pertinence and manner of applying Seltzer to claims 1-6, 9-14, 17-22 for which re-examination is requested is as follows:

Claims of '211 Patent	Seltzer
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Seltzer presents an LFS file system, with data stored in memory and on a disk storage system. *See e.g.* Seltzer, p. 59.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	Seltzer teaches that the LFS file system maintains an on-disk root inode that points directly and indirectly to a first set of storage blocks on disk in a consistent state. *See e.g.* Sec. 5.2.1.3 at p. 59, Sec. 6.1.1 at pp.70-71. This is further illustrated in Fig. 5-5:

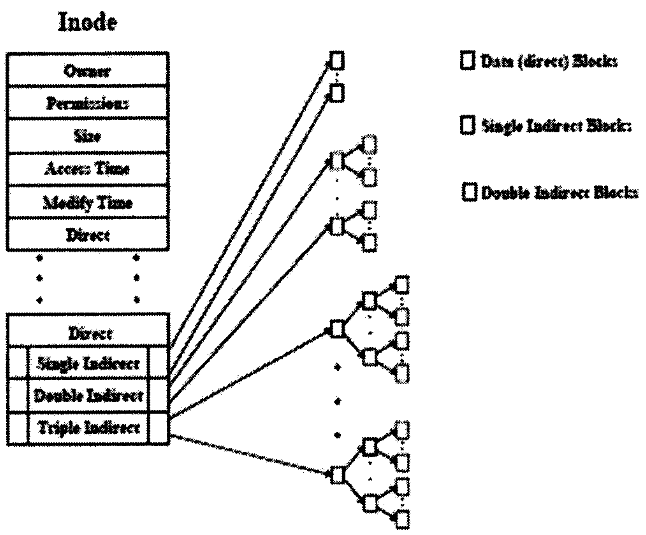
EM\7225346.3

	 Inode Owner Permissions Size Access Time Modify Time Direct Direct Single Indirect Double Indirect Triple Indirect Data (direct) Blocks Single Indirect Blocks Double Indirect Blocks Figure 5-5: File Index Structure (inode).
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Seltzer teaches an in-memory inode containing the physical representation of the file system layout. Seltzer, Sec. 5.2.1.3, p. 59, Fig. 5-5. The in-memory inode "additionally includes lists of buffers and links to other inodes. . . . The in-memory inode is extended to have a list of transaction-protected buffers in addition to its clean and dirty buffer lists." *Id.* "When a file is written, the new data blocks are appended to the log, and the index structure and indirect blocks are modified (in memory) to contain the new disk address of the newly written block." Seltzer, Sec. 4.1, pp. 31-32. Therefore, Seltzer teaches that the in-memory inode represents a consistent up-to-date state of the file system, pointing directly and indirectly to buffers containing new or modified data as well to the unchanged blocks of data. Seltzer further teaches that "[w]hen writing, LFS gathers many dirty pages and prepares to write them to disk sequentially in the next available segment. At that time, LFS sorts the blocks by logical block number, assigns them disk addresses, and updates the meta-data to reflect their addresses." Seltzer, Sec. 6.1.1, p. 71. Once some buffers holding dirty pages (changed data) are flushed to disk blocks, they, together with unchanged blocks extant on disk, constitute a second set of blocks. This second set of blocks, in combination with un-flushed dirty buffers, represent the up-to-date file system in a consistent state.

2. A method as in claim 1, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk. *See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.*
3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Seltzer teaches that the new *ifile* inode is flushed to disk after the dirty buffers are flushed. *See* Seltzer, Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88.
4. A method as in claim 3, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Seltzer teaches that the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Seltzer, Sec. 6.1.1, p. 71. In view of this teaching, it would have been obvious for one of ordinary skill in the art to replicate the root inode to recover from crashes that corrupt the primary copy of the root inode.
5. A method as in claim 1, further comprising the step of creating a snapshot of said file system by copying only said on-disk root inode.	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk. *See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.*
6. A method as in claim 5, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk. *See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.* The checkpoint inherently shares unmodified blocks with the active file system.
9. A device comprising: a processor; a memory; and	The file system taught in Seltzer is inherently executed on a processor with a memory.
a storage system including one or more hard disks;	Seltzer presents an LFS file system, with data stored in memory and on a disk storage system. *See e.g.* Seltzer, p. 59.
wherein said memory and said storage system store a file system; and	Seltzer presents an LFS file system, with data stored in memory and on a disk storage system. *See e.g.* Seltzer, p. 59.

wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Seltzer teaches that the LFS file system maintains an on-disk root inode that points directly and indirectly to a first set of storage blocks on disk in a consistent state. *See e.g.* Sec. 5.2.1.3 at p. 59, Sec. 6.1.1 at pp.70-71. This is further illustrated in Fig. 5-5:  Inode Owner Permissions Size Access Time Modify Time Direct Direct Single Indirect Double Indirect Triple Indirect Data (direct) Blocks Single Indirect Blocks Double Indirect Blocks **Figure 5-5: File Index Structure (inode).** Seltzer teaches an in-memory inode containing the physical representation of the file system layout. Seltzer, Sec. 5.2.1.3, p. 59, Fig. 5-5. The in-memory inode "additionally includes lists of buffers and links to other inodes. . . . The in-memory inode is extended to have a list of transaction-protected buffers in addition to its clean and dirty buffer lists." *Id.* "When a file is written, the new data blocks are appended to the log, and the index structure and indirect blocks are modified (in memory) to contain the new disk address of the newly written block." Seltzer, Sec. 4.1, pp. 31-32. Therefore, Seltzer teaches that the in-memory inode represents a consistent up-to-date state of the file system, pointing directly and indirectly to buffers containing new or modified data as well to the unchanged blocks of data. Seltzer further teaches that "[w]hen writing, LFS gathers many dirty pages and prepares to write them to disk sequentially in the next available segment. At that time, LFS sorts the blocks by logical block number, assigns them disk addresses, and updates the meta-data to reflect their addresses." Seltzer, Sec. 6.1.1, p. 71. Once

	some buffers holding dirty pages (changed data) are flushed to disk blocks, they, together with unchanged blocks extant on disk, constitute a second set of blocks. This second set of blocks, in combination with un-flushed dirty buffers, represent the up-to-date file system in a consistent state.
10. A device as in claim 9, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk. *See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.*
11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Seltzer teaches that the new *ifile* inode is flushed to disk after the dirty buffers are flushed. *See* Seltzer, Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88.
12. A device as in claim 11, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Seltzer teaches that the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Seltzer, Sec. 6.1.1, p. 71. In view of this teaching, it would have been obvious for one of ordinary skill in the art to replicate the root inode to recover from crashes that corrupt the primary copy of the root inode.
13. A device as in claim 9, wherein the instructions further comprise the step of creating a snapshot of said file system by copying only said on-disk root inode.	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk. *See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.*
14. A device as in claim 13, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk. *See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.* The checkpoint inherently shares unmodified blocks with the active file system.
17. An article of manufacture comprising a	The file system of Seltzer is inherently stored on

machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	machine readable medium in the form of instructions executable by a processor coupled to a memory and one or more hard disks. Seltzer teaches that the LFS file system maintains an on-disk root inode that points directly and indirectly to a first set of storage blocks on disk in a consistent state. *See e.g.* Sec. 5.2.1.3 at p. 59, Sec. 6.1.1 at pp.70-71. This is further illustrated in Fig. 5-5:  Inode Owner Permissions Size Access Time Modify Time Direct Direct Single Indirect Double Indirect Triple Indirect Data (direct) Blocks Single Indirect Blocks Double Indirect Blocks **Figure 5-5: File Index Structure (inode).** Seltzer teaches an in-memory inode containing the physical representation of the file system layout. Seltzer, Sec. 5.2.1.3, p. 59, Fig. 5-5. The in-memory inode "additionally includes lists of buffers and links to other inodes. . . . The in-memory inode is extended to have a list of transaction-protected buffers in addition to its clean and dirty buffer lists." *Id.* "When a file is written, the new data blocks are appended to the log, and the index structure and indirect blocks are modified (in memory) to contain the new disk address of the newly written block." Seltzer, Sec. 4.1, pp. 31-32. Therefore, Seltzer teaches that the in-memory inode represents a consistent up-to-date state of the file system, pointing directly and indirectly to buffers containing new or modified data as well to the unchanged blocks of data. Seltzer further teaches that "[w]hen writing, LFS gathers many dirty pages and prepares to write them to disk sequentially in

	the next available segment. At that time, LFS sorts the blocks by logical block number, assigns them disk addresses, and updates the meta-data to reflect their addresses." Seltzer, Sec. 6.1.1, p. 71. Once some buffers holding dirty pages (changed data) are flushed to disk blocks, they, together with unchanged blocks extant on disk, constitute a second set of blocks. This second set of blocks, in combination with un-flushed dirty buffers, represent the up-to-date file system in a consistent state.
18. An article of manufacture as in claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second consistent state.	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk. *See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.*
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Seltzer teaches that the new *ifile* inode is flushed to disk after the dirty buffers are flushed. *See* Seltzer, Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88.
20. An article of manufacture as in claim 19, wherein updating said on-disk root inode comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Seltzer teaches that the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Seltzer, Sec. 6.1.1, p. 71. In view of this teaching, it would have been obvious for one of ordinary skill in the art to replicate the root inode to recover from crashes that corrupt the primary copy of the root inode.
21. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create a snapshot of said file system by copying only said on-disk root inode.	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk. *See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.*
22. An article of manufacture as in claim 21, wherein the instructions cause the	Seltzer teaches that the LFS file system atomically advances when the *ifile* inode is committed to disk.

processor to create said snapshot so that said snapshot and said file system share said first set of blocks on said storage system when said snapshot is created.	*See e.g.* Seltzer Sec. 6.1.1, pp. 71-72, Sec. 6.3.3, p. 88. A consistent checkpoint is created when the *ifile* inode is flushed to disk. *Id.* The checkpoint inherently shares unmodified blocks with the active file system.

Fourth Basis of Invalidity

The reference applicable to the fourth basis of invalidity is:

1. Schilling, *Design and implementation of a fast file system for Unix with special consideration of technical parameters of optical storage media and multimedia applications*, Thesis submitted to Technical University of Berlin on 5/23/1991, translated from German. ("Schilling"). All pages cited are to the English translation.

The pertinence and manner of applying Schilling to claims 1-24 for which re-examination is requested is as follows:

Claims of '211 Patent	**Schilling**
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Schilling teaches a WoFS file system using a cache memory and one or more WORM optical disks. Schilling further teaches that the WoFS file system can be used with magnetic hard disks. Schilling, Sec. 1.2, p. 5.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	Schilling teaches that the WoFS file system is based on the well-known Unix file structure (as implemented in industry-standard BSD file system), which includes a superblock (root inode), inodes with levels of indirection, and data blocks. Schilling, Sec. 1.2.2, p. 6. WoFS modifies this structure to account for the write once – read many (WORM) nature of the optical media. *See e.g.* Schilling, Sec. 12.3-1.2.4. The modified system retains superblocks, inodes (referred to as *generation* nodes or *gnodes*) and data blocks. *Id.* These structures are present on the WORM disks. *See* Schilling Sec. 1.2.5. WoFS uses copy-on-write techniques to achieve file system consistency, writing new versions of files to a previously unused area of the medium. *See e.g.* Schilling, Sec. 1.2.5.1, p. 12
maintaining an incore root inode in said memory, said incore root inode pointing	Schilling teaches a cache memory where an in-core copy of the on-disk file system structure is

directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	maintained. *See* Schilling, Sec. 1.4.2. Changes to the file system occur in the cached file system structure, which thereby diverges from the on-disk structure. *See* Schilling, Sec. 1.2.7. It is inherent that changed data is stored in buffers in the cache before being written to disk. Schilling teaches that gnode updates are periodically forced to disk. Schilling, Sec. 1.2.7.4, p. 20. It is inherent that updated gnodes are buffered in the cache before being written to disk. Schilling further teaches that the superblock is written to disk after data blocks and intermediate gnodes. Schilling, Sec. 1.2.7.2, p. 19. It is inherent that the updated superblock, pointing to buffered and on-disk gnodes and data blocks, is buffered in the cache before being forced to disk.
2. A method as in claim 1, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Schilling's WoFS system advances atomically when the superblock update is forced to disk. *See* Schilling, Sec. 1.2.7.2.
3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Schilling's WoFS system advances atomically when the superblock update is forced to disk. *See* Schilling, Sec. 1.2.7.2. Schilling further teaches that the superblock is written to disk after data blocks and intermediate gnodes. Schilling, Sec. 1.2.7.2, p. 19.
4. A method as in claim 3, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. Schilling teaches that the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Schilling, Sec. 1.2.7.1, p. 19.
5. A method as in claim 1, further comprising the step of creating a snapshot of said file system by copying only said on-disk root inode.	Schilling teaches that old versions of gnodes and associated data (equivalent to snapshots) are retained because WORM disks cannot be erased. Schilling, Sec. 1.2.5.1, p. 12. Schilling further teaches accessing old versions through a directory. *Id.* It is inherent that the directory would be a part of the root node or superblock

	and be copied when the root is copied.
6. A method as in claim 5, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Due to the copy-on-write characteristic of the WoFS file system, older and newer versions of data would share unchanged data blocks.
7. A method as in claim 1, further comprising the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	Schilling teaches that old versions of gnodes and associated data (equivalent to snapshots) are retained because WORM disks cannot be erased. Schilling, Sec. 1.2.5.1, p. 12. Schilling further teaches accessing old versions through a directory. *Id.* It is inherent that the directory would be a part of the root node or superblock and be copied when the root is copied.
8. A method as in claim 7, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	Due to the copy-on-write characteristic of the WoFS file system, older and newer versions of data would share unchanged data blocks.
9. A device comprising: a processor; a memory; and	The file system taught in Schilling is inherently executed on a processor with a memory.
a storage system including one or more hard disks;	Schilling teaches a WoFS file system using a cache memory and one or more WORM optical disks. Schilling further teaches that the WoFS file system can be used with magnetic hard disks. Schilling, Sec. 1.2, p. 5
wherein said memory and said storage system store a file system; and	The file system in Schilling is stored in memory and on a storage system. *Id.*
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said	Schilling teaches that the WoFS file system is based on the well-known Unix file structure (as implemented in industry-standard BSD file system), which includes a superblock (root inode), inodes with levels of indirection, and data blocks. Schilling, Sec. 1.2.2, p. 6. WoFS modifies this structure to account for the write once – read many (WORM) nature of the optical media. *See e.g.* Schilling, Sec. 12.3-1.2.4. The modified system retains superblocks, inodes (referred to as *generation* nodes or *gnodes*) and data blocks. *Id.* These structures are present on the WORM disks. *See* Schilling Sec. 1.2.5. WoFS uses copy-on-write techniques to achieve file system

second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	consistency, writing new versions of files to a previously unused area of the medium. *See e.g.* Schilling, Sec. 1.2.5.1, p. 12 Schilling further teaches a cache memory where an in-core copy of the on-disk file system structure is maintained. *See* Schilling, Sec. 1.4.2. Changes to the file system occur in the cached file system structure, which thereby diverges from the on-disk structure. *See* Schilling, Sec. 1.2.7. It is inherent that changed data is stored in buffers in the cache before being written to disk. Schilling teaches that gnode updates are periodically forced to disk. Schilling, Sec. 1.2.7.4, p. 20. It is inherent that updated gnodes are buffered in the cache before being written to disk. Schilling further teaches that the superblock is written to disk after data blocks and intermediate gnodes. Schilling, Sec. 1.2.7.2, p. 19. It is inherent that the updated superblock, pointing to buffered and on-disk gnodes and data blocks, is buffered in the cache before being forced to disk.
10. A device as in claim 9, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Schilling's WoFS system advances atomically when the superblock update is forced to disk. *See* Schilling, Sec. 1.2.7.2.
11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Schilling's WoFS system advances atomically when the superblock update is forced to disk. *See* Schilling, Sec. 1.2.7.2. Schilling further teaches that the superblock is written to disk after data blocks and intermediate gnodes. Schilling, Sec. 1.2.7.2, p. 19.
12. A device as in claim 11, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. Schilling teaches that the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Schilling, Sec. 1.2.7.1, p. 19.

13. A device as in claim 9, wherein the instructions further comprise the step of creating a snapshot of said file system by copying only said on-disk root inode.	Schilling teaches that old versions of gnodes and associated data (equivalent to snapshots) are retained because WORM disks cannot be erased. Schilling, Sec. 1.2.5.1, p. 12. Schilling further teaches accessing old versions through a directory. *Id.* It is inherent that the directory would be a part of the root node or superblock and be copied when the root is copied.
14. A device as in claim 13, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Due to the copy-on-write characteristic of the WoFS file system, older and newer versions of data would share unchanged data blocks.
15. A device as in claim 9, wherein the instructions further comprise the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	Schilling teaches that old versions of gnodes and associated data (equivalent to snapshots) are retained because WORM disks cannot be erased. Schilling, Sec. 1.2.5.1, p. 12. Schilling further teaches accessing old versions through a directory. *Id.* It is inherent that the directory would be a part of the root node or superblock and be copied when the root is copied.
16. A device as in claim 15, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	Due to the copy-on-write characteristic of the WoFS file system, older and newer versions of data would share unchanged data blocks.
17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system,	The file system of Schilling is inherently stored on machine readable medium in the form of instructions executable by a processor coupled to a memory and one or more hard disks. Schilling teaches that the WoFS file system is based on the well-known Unix file structure (as implemented in industry-standard BSD file system), which includes a superblock (root inode), inodes with levels of indirection, and data blocks. Schilling, Sec. 1.2.2, p. 6. WoFS modifies this structure to account for the write once – read many (WORM) nature of the optical media. *See e.g.* Schilling, Sec. 12.3-1.2.4. The modified system retains superblocks, inodes (referred to as *generation* nodes or *gnodes*) and data blocks. *Id.* These

said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	structures are present on the WORM disks. *See* Schilling Sec. 1.2.5. WoFS uses copy-on-write techniques to achieve file system consistency, writing new versions of files to a previously unused area of the medium. *See e.g.* Schilling, Sec. 1.2.5.1, p. 12 Schilling further teaches a cache memory where an in-core copy of the on-disk file system structure is maintained. *See* Schilling, Sec. 1.4.2. Changes to the file system occur in the cached file system structure, which thereby diverges from the on-disk structure. *See* Schilling, Sec. 1.2.7. It is inherent that changed data is stored in buffers in the cache before being written to disk. Schilling teaches that gnode updates are periodically forced to disk. Schilling, Sec. 1.2.7.4, p. 20. It is inherent that updated gnodes are buffered in the cache before being written to disk. Schilling further teaches that the superblock is written to disk after data blocks and intermediate gnodes. Schilling, Sec. 1.2.7.2, p. 19. It is inherent that the updated superblock, pointing to buffered and on-disk gnodes and data blocks, is buffered in the cache before being forced to disk.
18. An article of manufacture as in claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second consistent state.	Schilling's WoFS system advances atomically when the superblock update is forced to disk. *See* Schilling, Sec. 1.2.7.2.
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Schilling's WoFS system advances atomically when the superblock update is forced to disk. *See* Schilling, Sec. 1.2.7.2. Schilling further teaches that the superblock is written to disk after data blocks and intermediate gnodes. Schilling, Sec. 1.2.7.2, p. 19.
20. An article of manufacture as in claim 19, wherein updating said on-disk root inode comprises updating said on-disk root inode and	Replicating the metadata, such as a root inode, is a technique that is well known in the art. Schilling teaches that the superblock is

then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Schilling, Sec. 1.2.7.1, p. 19.
21. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create a snapshot of said file system by copying only said on-disk root inode.	Schilling teaches that old versions of gnodes and associated data (equivalent to snapshots) are retained because WORM disks cannot be erased. Schilling, Sec. 1.2.5.1, p. 12. Schilling further teaches accessing old versions through a directory. *Id.* It is inherent that the directory would be a part of the root node or superblock and be copied when the root is copied.
22. An article of manufacture as in claim 21, wherein the instructions cause the processor to create said snapshot so that said snapshot and said file system share said first set of blocks on said storage system when said snapshot is created.	Due to the copy-on-write characteristic of the WoFS file system, older and newer versions of data would share unchanged data blocks.
23. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create plural snapshots of said file system by copying only said on-disk root inode at different times.	Schilling teaches that old versions of gnodes and associated data (equivalent to snapshots) are retained because WORM disks cannot be erased. Schilling, Sec. 1.2.5.1, p. 12. Schilling further teaches accessing old versions through a directory. *Id.* It is inherent that the directory would be a part of the root node or superblock and be copied when the root is copied.
24. An article of manufacture as in claim 23, wherein the instructions, when executed by the processor, cause the processor to create each one of said plural snapshots so that each one of said snapshots and said file system share said first set of blocks on said storage system when each one of said plural snapshots is created.	Due to the copy-on-write characteristic of the WoFS file system, older and newer versions of data would share unchanged data blocks.

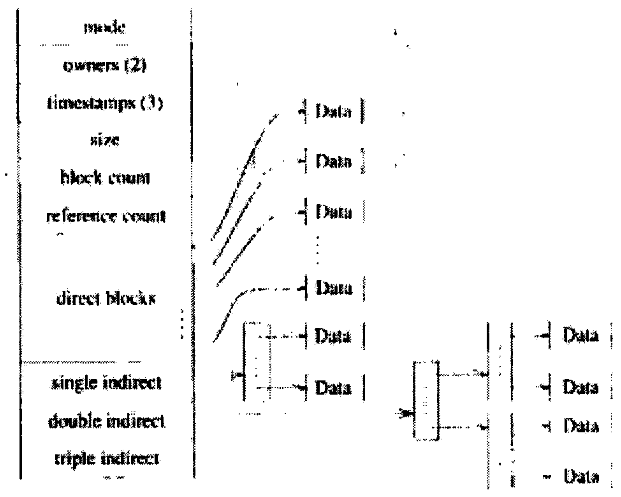
Fifth Basis of Invalidity

The references applicable to the fifth basis of invalidity are:

1. Leffler, McKusick, et. al., *4.3BSD Unix Operating System*, Addison-Wesley Publishing Co., 1990 ("Leffler").

2. Bach, *The Design of the Unix Operating System*, Prentice Hall, 1990 ("Bach").

The pertinence and manner of applying Leffler and Bach to claims 1-4, 9-12, 17-20 for which re-examination is requested is as follows:

Claims of '211 Patent	Leffler and Bach
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Leffler teaches an implementation of the well-known Unix file system, storing information in memory and on hard disk. *See* Leffler, Ch. 7.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	Leffler teaches maintaining an on-disk root inode pointing directly and indirectly to a set of blocks on disk in a consistent state: **Figure 7.6** The structure of an inode.  mode owners (2) timestamps (3) size block count reference count direct blocks single indirect double indirect triple indirect Data *See* Leffler Sec. 7.2
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Leffler teaches that a copy of the root inode (inode describing the root directory) and copies of lower level inodes and data blocks are maintained in memory (in-core). Leffler, Sec. 7.4, pp. 203-07 When the system writes new data, it allocates buffers in memory for the new data. *See e.g.* Leffler, Sec. 7.5, pp. 211-12. Periodically, the buffers are flushed to disk during a *sync* process. *See e.g.* Leffler, Sec. 7.4, p. 207. It is inherent that the incore root inode is updated to point to the new buffers and blocks. For instance, Bach (a conventional well known Unix textbook) teaches that in Unix, the in-core inode is updated when data blocks change. Bach, Sec. 4.1, pp. 62-63.